

# RIOTS + AD: Integrating Automatic Differentiation into Computational Optimal Control

YangQuan Chen

Department of Electrical and Computer Engineering  
Utah State University  
Logan, Utah 84322-4160, USA  
yqchen@ieee.org

Jason Gu

Department of Electrical and Computer Engineering  
Dalhousie University  
Halifax, Nova Scotia, Canada B3J 2X4  
jason.gu@ieee.org

Jinsong Liang

Department of Mechanical and Aerospace Engineering  
Utah State University  
Logan, Utah 84322-4130, USA  
jsliang@ieee.org

Rees Fullmer

Department of Mechanical and Aerospace Engineering  
Utah State University  
Logan, Utah 84322-4130, USA  
rfullmer@engineering.usu.edu

**Abstract**—RIOTS<sup>1</sup> is a toolbox for Matlab<sup>2</sup>, written mostly in C, Fortran and M-file scripts, that provides an interactive environment for solving a very broad class of optimal control problems (OCPs). Automatic differentiation (AD) can numerically compute fast and accurately the derivatives of a function in general form that can be represented by a piece of code (Matlab or C/C++). The integration of AD makes RIOTS a more powerful and user-friendly tool for computational optimal control. This paper describes the RIOTS toolbox, some AD tools and our efforts in integrating AD into RIOTS. Some comparisons are made to show that AD is a useful function for RIOTS.

**Index Terms**—Optimal control problem solver, automatic differentiation, consistent approximation, B-splines, Runge-Kutta integration.

## I. INTRODUCTION

Optimization is a routine work in engineering practice. In general, optimization tasks can be classified into two categories: static optimization tasks and dynamic ones. Dynamic optimization has not been developed as “matured” as static optimization. More often, dynamic optimization is referred to as “optimal control problems (OCPs)”. Numerical methods for solving optimal control problems have not reached the stage that, say, methods for solving differential equations have reached. Solving an optimal control problem can, depending on the difficulty of the problem, require significant user involvement in the solution process. This sometimes requires the user to understand the theory of optimal control, optimization and/or numerical approximation methods. Among the existing software packages for numerically solving dynamic optimization or optimal control problems, such as SOCS [1], RIOTS [2], DIRCOL [3], or MISER3 [4], no single program can solve all sort of problems. For a recent survey on solving OCPs, refer to [5].

Nevertheless, theoretically speaking, it is well known that dynamic programming (DP) of Bellman can solve all types

of OCPs [6], [7], [8]. It is a powerful method for solving optimization problems by breaking up a complex optimization problem into a number of simpler problems. The solution of the simpler problems leads to the solution of the original problem. This is an attractive feature with guaranteed global optimum. Although the computers are now more and more powerful, use of dynamic programming idea to solving OCPs is still not popular today due to the inherent drawbacks of DP method: curse of dimensionality, problems in the expanding grids and problems in the interpolations etc., that limit its use to only solving problems of very low dimension.

In view of the above discussions, while solving OCPs is still an “art”, it is important to have a good software environment for the user to play with. Ideally, no compiling/linking is required. To this regard, RIOTS is the most favorable due to the Matlab platform. RIOTS is designed as a Matlab toolbox written mostly in C, Fortran and M-file scripts. It provides an interactive environment for solving a very broad class of optimal control problems. RIOTS runs under Windows 98/2000/XP or Linux operating systems. The user-OCPs can be prepared purely in M-files and no compiler is needed to solve the OCPs. To speed up the OCP solving process, there are two ways to proceed: by using the Matlab Compiler or by providing the user-OCP in C which is to be compiled by a C-compiler and then linked with some pre-built linking libraries (currently, Microsoft Visual C++ and GNU GCC are supported, for Windows and Linux, respectively).

To use RIOTS, the user needs to provide the first order derivative with respect to state variables and control variables of the functions describing the optimal control problem. For some complicated problems, the derivatives are hard, if not impossible, to obtain manually. If not provided by the user, the derivatives can still be calculated by RIOTS using the numerical differentiation, which is slow and inaccurate. Automatic differentiation (AD) [9],[10] is a technique to solve the above problems. AD is able to compute any order derivatives of a function in general form that can be represented by a piece of code (Matlab, Fortran, or C/C++) fast and accurately. Two different AD interfaces, ADOL-C [11] and ADMAT [12], have been added to RIOTS, making RIOTS a more powerful and user-friendly toolbox for solving optimal control problems<sup>3</sup>.

August 2003. For submission to The 2003 International Conference on Control Science and Engineering, December 18-20, 2003, Harbin, China (ICCSE2003). <http://www.ee.cuhk.edu.hk/~qhmeng/iccse/>  
Corresponding author: Dr YangQuan Chen. E-mail: yqchen@ece.usu.edu; Tel. 01-435-7970148; Fax: 01-435-7973054. URL: <http://www.csois.usu.edu/people/yqchen>.

<sup>1</sup><http://schwartz-home.com/~adam/RIOTS>

<sup>2</sup>Matlab is a registered trademark of Mathworks, Inc. (<http://www.mathworks.com/>)

<sup>3</sup>Integrating ADOL-C with RIOTS is still in alpha phase.

The paper is organized as follows. In Sec. II, we introduce the main features of RIOTS. In Sec. III, we introduce briefly AD and the available AD implementations for use with RIOTS, mainly ADMAT and ADOL-C. In Sec. IV, the main results integrating AD in RIOTS are presented. Section V concludes the paper.

## II. RIOTS: A GENERAL OPTIMAL CONTROL PROBLEM SOLVER

RIOTS is a group of programs and utilities, designed as a toolbox for Matlab. The numerical methods used by RIOTS are supported by the theory in [13], which uses the approach of consistent approximations as defined in [14]. In this approach, a solution is obtained as an accumulation point of the solutions to a sequence of discrete-time optimal control problems that are, in a specific sense, consistent approximations to the original continuous-time, optimal control problem. The discrete-time optimal control problems are constructed by discretizing the system dynamics with one of four fixed step-size Runge-Kutta integration methods and by representing the controls as finite-dimensional B-splines. Note that RIOTS also includes a variable step-size integration routine and a discrete-time solver. The integration proceeds on a (possibly non-uniform) mesh that specifies the splines breakpoints. The solution obtained for one such discretized problem can be used to select a new integration mesh upon which the optimal control problem can be re-discretized to produce a new discrete-time problem that more accurately approximates the original problem. In practice, only a few such re-discretizations need to be performed to achieve an acceptable solution.

RIOTS provides three different programs that perform the discretization and solve the finite-dimensional discrete-time problem. The appropriate choice of optimization program depends on the type of problem being solved as well as the number of points in the integration mesh. In addition to these optimization programs, RIOTS also includes other utility programs that are used to refine the discretization mesh, to compute estimates of integration errors, to compute estimates for the error between the numerically obtained solution and the optimal control and to deal with oscillations that arise in the numerical solution of singular optimal control problems.

### A. Major features of RIOTS

The name RIOTS stands for “**R**ecursive<sup>4</sup> **I**ntegration **O**ptimal **T**rajectory **S**olver.” This name highlights the fact that the function values and gradients needed to find the optimal solutions are computed by forward and backward integration of certain differential equations.

RIOTS<sup>5</sup> is a collection of programs that are callable from the mathematical simulation program Matlab for Windows and Linux. Most of these programs are written in either C, Fortran (and linked into Matlab using Matlab’s MEX/DLL facility) or Matlab’s M-script language. All of Matlab’s functionality, including command line execution and data entry and data plotting, are available to the user. The following is a list of some of the main features of RIOTS.

<sup>4</sup>*Iterative* is more accurate but would not lead to a nice acronym.

<sup>5</sup>RIOTS runs in Matlab versions 4.x, 5.x, 6.x. The Matlab Splines Toolbox is required. Evaluation versions are downloadable from <http://www.accesscom.com/~adam/RIOTS/>

TABLE I  
THREE CATEGORIES OF PROGRAMS IN RIOTS

Simulation Routines	Optimization Routines	Utility Programs
<code>simulate</code>	<code>riots</code>	<code>control_error</code>
<code>check_deriv</code>	<code>pdmin</code>	<code>distribute</code>
<code>check_grad</code>	<code>aug_lagrng</code>	<code>est_error</code>
<code>eval_fnc</code>	<code>outer</code>	<code>make_spline</code>
		<code>transform</code>

- Solves a very large class of finite-time optimal controls problems that includes: trajectory and endpoint constraints, control bounds, variable initial conditions (free final time problems), and problems with integral and/or endpoint cost functions.
- System functions can be supplied by the user as either object code or M-files.
- System dynamics can be integrated with fixed step-size Runge-Kutta integration, a discrete-time solver or a variable step-size method. The software automatically computes gradients for all functions with respect to the controls and any free initial conditions. These gradients are computed exactly for the fixed step-size routines.
- The controls are represented as splines. This allows for a high degree of function approximation accuracy without requiring a large number of control parameters.
- The optimization routines use a coordinate transformation that creates an orthonormal basis for the splines subspace of controls. The use of an orthogonal basis can result in a significant reduction in the number of iterations required to solve a problem and an increase in the solution accuracy. It also makes the termination tests independent of the discretization level.
- There are three main optimization routines, each suited for different levels of generality of the optimal control problem. The most general is based on sequential quadratic programming methods. The most restrictive, but most efficient for large discretization levels, is based on the projected descent method. A third algorithm uses the projected descent method in conjunction with an augmented Lagrangian formulation.
- There are programs that provide estimates of the integration error for the fixed step-size Runge-Kutta methods and estimates of the error of the numerically obtained optimal control.
- The main optimization routine includes a special feature for dealing with singular optimal control problems.
- The algorithms are all founded on rigorous convergence theory.

In addition to being able to accurately and efficiently solve a broad class of optimal control problems, RIOTS is designed in a modular, toolbox fashion that allows the user to experiment with the optimal control algorithms and construct new algorithms. The programs `outer` and `aug_lagrng`, described in detail in [2], are examples of this toolbox approach to constructing algorithms.

RIOTS is a collection of several different programs (including a program which is, itself, called `riots`) that fall into roughly three categories: integration/simulation routines, optimization routines, and utility programs. Of these programs, the ones available to the user are listed in Tab. I

Several of the programs in RIOTS require functions that are available in the Matlab Splines toolbox. In addition to these programs, the user must also supply a set of routines that describe the optimal control problem which must be solved. Several example optimal control problems come supplied with RIOTS. Finally, there is a Matlab script called RIOTSdem which provides a demonstration of some of the main features of RIOTS.

### B. Optimal control problems in RIOTS

RIOTS is designed to solve optimal control problem of the form

$$\mathbf{OCP} : \min_{(u, \xi) \in L_{\infty}^m[a, b] \times R^n} \left\{ f(u, \xi) \doteq g_o(\xi, x(b)) + \int_a^b l_o(t, x, u) dt \right\}$$

subject to:

$$\begin{aligned} \dot{x} &= h(t, x, u), \quad x(a) = \xi, \quad t \in [a, b], \\ u_{min}^j(t) &\leq u^j(t) \leq u_{max}^j(t), \quad j = 1, \dots, m, \\ \xi_{min}^j &\leq \xi^j \leq \xi_{max}^j, \quad j = 1, \dots, n, \\ l_{ti}^{\nu}(t, x(t), u(t)) &\leq 0, \quad \nu \in \mathbf{q}_{ti}, \quad t \in [a, b], \\ g_{ei}^{\nu}(\xi, x(b)) &\leq 0, \quad \nu \in \mathbf{q}_{ei}, \\ g_{ee}^{\nu}(\xi, x(b)) &= 0, \quad \nu \in \mathbf{q}_{ee}, \end{aligned}$$

where  $x(t) \in R^n$ ,  $u(t) \in R^m$ ,  $g : R^n \times R^n \rightarrow R$ ,  $l : R \times R^n \times R^m \rightarrow R$ ,  $h : R \times R^n \times R^m \rightarrow R^n$  and we have used the notation  $\mathbf{q} \doteq 1, \dots, q$  and  $L_{\infty}^m[a, b]$  is the space of Lebesgue measurable, essentially bounded functions  $[a, b] \rightarrow R^m$ . The functions in **OCP** can also depend upon parameters which are passed from Matlab at execution time using `get_flags`. Refer to [2] for details.

The subscripts *o*, *ti*, *ei*, and *ee* on the functions  $g(\cdot, \cdot)$  and  $l(\cdot, \cdot, \cdot)$  stand for, respectively, ‘‘objective function’’, ‘‘trajectory constraint’’, ‘‘endpoint inequality constraint’’ and ‘‘endpoint equality constraint’’. The subscripts for  $g(\cdot, \cdot)$  and  $l(\cdot, \cdot, \cdot)$  are omitted when all functions are being considered without regard to the subscript. The functions in the description of problem **OCP**, and the derivatives of these functions<sup>6</sup>, must be supplied by the user as either object code or as M-files. The bounds on the components of  $x_i$  and  $u$  are specified on the Matlab command line at run-time.

The optimal control problem **OCP** allows optimization over both the control  $u$  and one or more of the initial states  $\xi$ . To be concise, we will define the variable

$$\eta = (u, \xi) \in H_2 \doteq L_{\infty}^m[a, b] \times R^n.$$

With this notation, we can write, for example,  $f(\eta)$  instead of  $f(\xi, u)$ . We define the inner product on  $H_2$  as

$$\langle \eta_1, \eta_2 \rangle_{H_2} \doteq \langle u_1, u_2 \rangle_{L_2} + \langle \xi_1, \xi_2 \rangle.$$

The norm corresponding to this inner product is given by  $\|\eta\|_{H_2} = \langle \eta, \eta \rangle_{H_2}^{1/2}$ . Note that  $H_2$  is a pre-Hilbert space.

<sup>6</sup>If the user does not supply derivatives, the problem can still be solved using `riots` with finite-difference computation of the gradients.

1) *Transcription for Free Final Time Problems*: Problem **OCP** is a fixed final time optimal control problem. However, free final time problems are easily incorporated into the form of **OCP** by augmenting the system dynamics with two additional states (one additional state for autonomous problems). The idea is to specify a nominal time interval,  $[a, b]$ , for the problem and to use a scale factor, adjustable by the optimization procedure, to scale the system dynamics and hence, in effect, scale the duration of the time interval. This scale factor, and the scaled time, are represented by the extra states. Then RIOTS can minimize over the initial value of the extra states to adjust the scaling. For example, the free final time optimal control problem

$$\min_{u, T} \tilde{g}(T, y(T)) + \int_a^{a+T} \tilde{l}(t, y, u) dt$$

$$\text{subject to } \dot{y} = \tilde{h}(t, y, u), y(a) = \zeta, t \in [a, a+T],$$

can, with an augmented state vector  $x \doteq (y, x^{n-1}, x^n)$ , be converted into the equivalent fixed final time optimal control problem

$$\begin{aligned} \min_{u, \xi^n} g(\xi, x(b)) + \int_a^b l(t, x, u) dt \\ \text{subject to } \dot{x} = h(t, x, u) \doteq \begin{pmatrix} x^n \tilde{h}(x^{n-1}, y, u) \\ x^n \\ 0 \end{pmatrix}, \\ x(a) = \xi = \begin{pmatrix} \zeta \\ a \\ \xi^n \end{pmatrix}, \quad t \in [a, b], \end{aligned}$$

where  $y$  is the first  $n-2$  components of  $x$ ,  $g(\xi, x(b)) \doteq \tilde{g}(a+T\xi^n, y(b))$ ,  $l(t, x, u) \doteq \tilde{l}(x^{n-1}, y, u)$  and  $b = a+T$ . Endpoint and trajectory constraints can be handled in the same way. The quantity  $T = b-a$  is the nominal trajectory duration. In this transcription,  $x^{n-1}$  plays the role of time and  $\xi^n$  is the *duration scale factor* so named because  $T\xi^n$  is the effective duration of the trajectories for the scaled dynamics. Thus, for any  $t \in [a, b]$ ,  $x^n(t) = \xi^n$ ,  $x^{n-1}(t) = a + (t-a)\xi^n$  and the solution,  $t_f$ , for the final time is  $t_f = x^{n-1}(b) = a + (b-a)\xi^n$ . Thus, the optimal duration is  $T^* = t_f - a = (b-a)\xi^n = T\xi^n$ . If  $a = 0$  and  $b = 1$ , then  $t_f = T^* = \xi^n$ . The main disadvantage to this transcription is that it converts linear systems into nonlinear systems.

For autonomous systems, the extra variable  $x^{n-1}$  is not needed. Note that, it is possible, even for non-autonomous systems, to transcribe minimum time problems into the form of **OCP** using only one extra state variable. However, this would require functions like  $h(t, x, u) = \tilde{h}(tx^n, y, u)$ . Since RIOTS does not expect the user to supply derivatives with respect to the  $t$  argument it can not properly compute derivatives for such functions. Hence, in the current implementation of RIOTS, the extra variable  $x^{n-1}$  is needed when transcribing non-autonomous, free final time problems.

2) *Trajectory constraints* : The definition of problem **OCP** allows trajectory constraints of the form  $l_{ti}(t, x, u) \leq 0$  to be handled directly. However, constraints of this form are quite burdensome computationally. This is mainly due to the fact that a separate gradient calculation must be performed for each point at which the trajectory constraint is evaluated.

At the expense of increased constraint violation, reduced solution accuracy and an increase in the number of iterations

required to obtain solutions, trajectory constraints can be converted into endpoint constraints which are computationally much easier to handle. This is accomplished as follows. The system is augmented with an extra state variable  $x^{n+1}$  with

$$\begin{aligned}\dot{x}^{n+1}(t) &= \mu \max\{0, l_{ti}(t, x(t), u(t))\}^2, \\ x^{n+1}(a) &= 0,\end{aligned}$$

where  $\mu > 0$  is a positive scalar. The right-hand side is squared so that it is differentiable with respect to  $x$  and  $u$ . Then it is clear that either of the endpoint constraints

$$\begin{aligned}g_{ei}(\xi, x(b)) &\doteq x^{n+1}(b) \leq 0 \quad \text{or} \\ g_{ee}(\xi, x(b)) &\doteq x^{n+1}(b) = 0\end{aligned}$$

is satisfied if and only if the original trajectory constraint is satisfied. In practice, the accuracy to which **OCP** can be solved with these endpoint constraints is quite limited because these endpoint constraints do not satisfy the standard constraint qualification [13]. This difficulty can be circumvented by eliminating the constraints altogether and, instead, adding to the objective function the penalty term  $g_o(\xi, x(b)) \doteq x^{n+1}(b)$  where now  $\mu$  serves as a penalty parameter.

However, in this approach,  $\mu$  must now be a large positive number and this will adversely affect the conditioning of the problem.

3) *Continuum Objective Functions and Minimax Problems*: Objective functions of the form

$$\min_u \max_{t \in [a,b]} l(t, x(t), u(t))$$

can be converted into the form used in problem **OCP** by augmenting the state vector with an additional state,  $w$ , such that

$$\dot{w} = 0; \quad w(0) = \xi^{n+1}$$

and forming the equivalent trajectory constrained problem

$$\min_{(u, \xi^{n+1})} \xi^{n+1}$$

subject to

$$l(t, x(t), u(t)) - \xi^{n+1} \leq 0, \quad t \in [a, b].$$

A similar transcription works for standard min-max objective functions of the form

$$\min_u \max_{\nu \in \mathbf{q}_o} g^\nu(u, \xi) + \int_a^b l^\nu(t, x(t), u(t)) dt.$$

In this case, an equivalent endpoint constrained problem with a single objective function,

$$\min_{u, \xi^{n+1}} \xi^{n+1}$$

subject to

$$\tilde{g}^\nu(u, \xi) - \xi^{n+1} \leq 0, \quad \nu \in \mathbf{q}_o$$

is formed by using the augmented state vector  $(x, w, z)$  with

$$\dot{w} = 0, \quad w(0) = \xi^{n+1}$$

$$\dot{z}^\nu = l^\nu(t, x(t), u(t)), \quad z^\nu(0) = 0, \quad \nu \in \mathbf{q}_o,$$

defining

$$\tilde{g}^\nu(u, \xi) \doteq g^\nu(u, \xi) + z^\nu(b).$$

### C. An example: bang-bang OCP

This is a textbook OCP problem [15] with control bounds and free final time. It is used to demonstrate the transcription of a free final time problem into a fixed final time problem. The transcribed problem has bounds on the control and free initial states. Also, **distribute.m** (see [13]) is used to improve integration mesh after an initial solution is found. A more accurate solution will then be computed by re-solving the problem on the new mesh. For self-containing, this OCP is given as follows:

$$\text{Problem Bang:} \quad \min_{u, T} J(u, T) = T$$

subject to

$$\dot{x}_1 = x_2; \quad x_1(0) = 0, x_1(T) = 300$$

$$\dot{x}_2 = u; \quad x_2(0) = 0, x_2(T) = 0$$

and

$$-2 \leq u(t) \leq 1, \quad \forall t \in [0, T].$$

This problem has an analytical solution which is given by  $T^* = 30$ . When  $t \in [0, 20)$ ,  $u^*(t) = 1$ ,  $x_1^*(t) = t^2/2$ ,  $x_2^*(t) = t$  while when  $t \in [20, 30]$ ,  $u^*(t) = -2$ ,  $x_1^*(t) = -t^2 + 60t - 600$ ,  $x_2^*(t) = 60 - 2t$ .

The above OCP is a minimum-time problem with two states and one input. This problem is converted into a fixed final time problem using the transcription described in Sec. 3. Only one extra state variable was needed since the problem has time-independent (autonomous) dynamics. First we define the integration mesh and then the initial conditions.

```
>> N = 20;           % Discretization level
>> T = 10;           % Nominal final time
>> t=[0:T/N:T];     % Nominal time interval
```

The nominal time interval is of duration T. Next, we specify a value for  $\xi^3$ , the duration scale factor, which is the initial condition for the augmented state. The quantity  $T\xi^3$  represents our guess for the optimal duration of the maneuver.

```
>> x0=[0 0 1]';     % Init. cond. for aug. sys.
>> fixed=[1 1 0]';  % with init. cond. fixed
>> x0_lower=[0 0 .1]'; %free init.cond. low bnd
>> x0_upper=[0 0 10]'; %free init.cond. up bnd
>> X0=[x0, fixed, x0_lower, x0_upper]
X0 =
     0     1.0000     0     0
     0     1.0000     0     0
  1.0000     0     0.1000  10.0000
```

The first column of X0 is the initial conditions for the problem; there are three states including the augmented state. The initial conditions for the original problem were  $x(0) = (0, 0)^T$ . The initial condition for the augmented state is set to  $x_0(3) = \xi^3 = 1$  to indicate that our initial guess for the optimal final time is one times the nominal final time of  $T = 10$ , i.e.  $\xi^3 T$ . The second column of X0 indicates which initial conditions are to be considered fixed and which are to be treated as free variables for the optimization program to adjust. A one indicates fixed and a zero indicates free. The third and fourth columns provide lower and upper bound for the free initial conditions.

```
>> u0=zeros(1, N+1); % f=x(3,1)=x0(3)
>> [u, x, f]=riots(X0, u0, t, -2, 1, [], 100, 2);
>> f*T           % Show the final time.
```

```
ans =
    29.9813
```

In this call to `riots`, we have also specified a lower bound of -2 and an upper bound of 1 for all of the control splines coefficients. Since we are using second order splines, this is equivalent to specifying bounds on the value of the control at the splines breakpoints, i.e. bounds on  $u(t_k)$ . We also specify that the second order Runge-Kutta integration routine should be used. The objective value  $f = \xi^3$  is the duration scale factor. The final time is given by  $a + (b - a)\xi^3 = T\xi^3 = 10f$ . Here we see that the final time is 29.9813. A plot of the control solution indicates a fairly broad transition region whereas we expect a bang-bang solution. We can try to improve the solution by redistributing the integration mesh. We can then re-solve the problem using the new mesh and starting from the previous solution interpolated onto the new mesh. This new mesh is stored in `new_t`, and `new_u` contains the control solution interpolated onto this new mesh.

```
>> [new_t,new_u]=distribut(t,u,x,2,[],1,1);

redistribute_factor = 7.0711
Redistributing mesh.

>> X0(:,1) = x(:,1);
>> [u,x,f]=riots(X0,new_u,new_t,-2,1,[],100,2);
>> f*10
ans =
    30.0000
```

Notice that before calling `riots` the second time, we set the initial conditions (the first column of `X0`) to `x(:,1)`, the first column of the trajectory solution returned from the preceding call to `riots`. Because  $\xi^3$  is a free variable in the optimization, `x(3,1)` is different than what was initially specified for `x0(3)`. Since `x(3,1)` is likely to be closer to the optimal value for  $\xi^3$  than our original guess we set the current guess for `X0(3,1)` to `x(3,1)`. We can see the improvement in the control solution and the solution for the final time. The reported final time solution is 30 and this happens to be the **exact answer**. The plot of the control solution before and after the mesh redistribution is shown below. The circles indicate where the mesh points are located. The improved solution does appear to be a bang-bang solution.

### III. A BRIEF INTRODUCTION TO AUTOMATIC DIFFERENTIATION

RIOTS needs the first order derivative of user-provided functions `h`, `g` and `tt` [2]. These derivatives can be provided either by the user or by RIOTS itself using the numerical finite difference approach. Both methods have their limitations. For some complicated optimal control problems, the derivatives of user-provided functions can be very hard, if not impossible, to obtain while the approximate derivatives calculated through the finite difference are inaccurate and computationally slow. AD can be used to solve the above problems.

As a relatively new technology, AD is able to compute derivatives of a general function numerically of any order. Although finite difference and symbolic differentiation can also be used to obtain the derivatives of functions, both

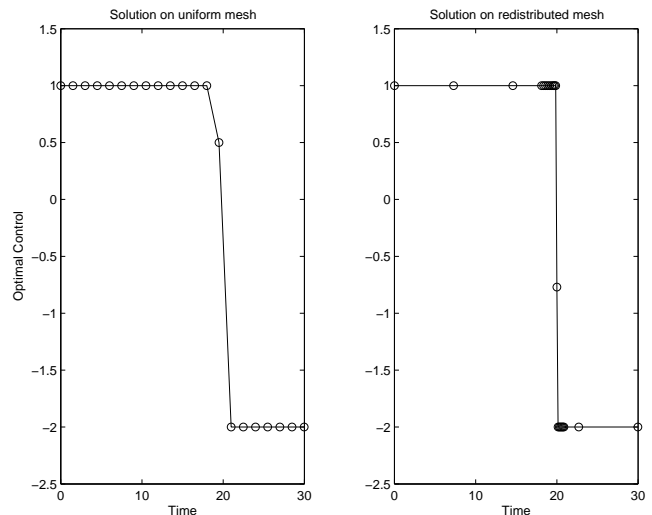


Fig. 1. Bang-bang OCP solutions with and without mesh re-distribution

methods are slow. Furthermore, the finite difference method gives only the approximate values of the derivatives while the symbolic differentiation method is incapable of computing the derivatives of complicated functions, such as the functions including the logic switches or case branching. AD, on the other hand, can be used to obtain the derivatives of general functions without the above limitations. The results from AD are accurate and the process is relatively fast. The basic idea is that, any general function, no matter how complicated, can be composed of some basic functions and operators. If the derivatives of these basic functions are known, the derivative of the general functions can be computed using the chain rule. For more information on AD, refer to [9][10]. For the comparison between symbolic differentiation and AD, refer to [16].

Currently there are several AD implementations available, including ADOL-C for C/C++ functions [11], ADIFOR for FORTRAN [17], and ADMIT/ADMAT [18][12] for Matlab. To avoid starting from scratch, we chose to make use of these available AD tools.

Since RIOTS accepts the user-OCPs in both the Matlab M-file format and the C-file format, to integrate AD into RIOTS, two sets of AD tools, for M-file and C-file respectively, are needed. For Matlab M-files, only ADMAT is available. For C-files, there are two candidates, namely ADMIT and ADOL-C. ADMIT computes the derivative by calling a set of Matlab M-files. So, to use ADMIT for C-file input, M-functions will be called during the C execution process. Due to the interpretative nature of the Matlab environment, calling M-files within C-files will severely slow down the speed of C execution, which is unacceptable, because the high speed is one of the main advantages using C-file interface instead of M-file interface. So, we choose ADOL-C, a pure C/C++ implementation, to integrate into RIOTS for C-file interface.

### IV. WITH AND WITHOUT AD: SOME COMPARISONS

We chose four optimal control problems to test the robustness and the speed, two main concerns to using AD. For detailed description of the four testing problems, refer to [2].

TABLE II  
EXECUTION SPEED COMPARISON FOR ADOL-C (SEC.)

Problem	User-provided Deriv.	Finite Diff.	ADOL-C
Rayleigh	0.03	0.12	xx
Bang	0.06	0.21	xx
Goddard	0.09	1.25	xx
Boeing	N/A	2.80	xx

TABLE III  
EXECUTION SPEED COMPARISON FOR ADMAT (SEC.)

Problem	User-provided Deriv.	Finite Diff.	ADMAT
Rayleigh	0.8	15.1	52.4
Bang	0.9	2.7	24.7
Goddard	6.2	219.8	2272.9
Boeing	N/A	19.5	178.9

ADOL-C passed all four tests smoothly, showing its good robustness. While ADMAT also passed all the tests, some manual adjustments of the formula expressions are necessary. For example, for the following Matlab functions,

```
function y=foo(x)
y = 0;
```

ADMAT failed to compute  $dy/dx$ , when  $x$  is a vector. Instead, the function has to be written as

```
function y=foo(x)
y=0*x(1)*x(2)
```

Comparison of execution speed using user-provided derivatives (C-files), finite difference, and ADOL-C is shown in Table II<sup>7</sup>.

From Table II, we can expect that although ADOL-C will be a bit slower than using the user-provided derivatives but it will (and should) be faster than the finite difference method for all the testing optimal control problems. We tend to conclude that for the complicated optimal control problems, in which the derivatives of user-provided functions are very hard, very time consuming and very error-prone, to derive manually, ADOL-C can replace finite difference completely without any performance sacrifice. Moreover, ADOL-C can be used to double check if the user-derived derivative routines are correct or not by using the RIOTS utility functions `check_deriv` and `check_grad`.

The comparison of execution speed using user-provided derivatives (M-files), finite difference, and ADMAT is shown in Table III.

From Table III, we can see that ADMAT is much slower than the other two. Combined with its non-robustness, it shows that integrating ADMAT into RIOTS is not a very successful practice. The reasons for the slowness are not completely clear to us. The most possible reason is the software bugs in the implementation of AD making use of the operator overloading technology in Matlab. We are expecting the further improvement of ADMAT. However, as a last remark, again, ADMAT interface with RIOTS is still useful in the sense that this extra AD functionality of RIOTS, although slow, can be used to double check if the user-derived derivative routines are correct or not by using the RIOTS utility functions `check_deriv` and `check_grad`.

<sup>7</sup>The accurate timing for ADOL-C will be provided in the final version of this paper when the ADOL-C interface to RIOTS will be fully tested.

## V. CONCLUDING REMARKS

The main features of RIOTS and the results of its integration with AD are introduced in this paper. To our best knowledge, RIOTS is the first software package for optimal control problem solution integrated with AD tools. AD tools significantly simplify the user-supplied files and improve the execution speed of optimization process. We believe that in the near future, AD is a must-have feature for all software packages for numerical optimal control.

## VI. REFERENCES

- [1] J. T. Betts, "SOCS: the sparse optimal control software family," <http://www.boeing.com/assocproducts/socs/index.html>, Tech. Rep., 1996.
- [2] A. Schwartz, E. Polak, and Y. Q. Chen, *RIOTS: a MATLAB toolbox for solving optimal control problems*, 1997. [Online]. Available: <http://www.accesscom.com/~adam/RIOTS>
- [3] O. V. Stryk, "DIRCOL: a direct collocation method for the numerical solution of optimal control problems," <http://www-m2.ma.tum.de/~stryk/dircol.html>, Tech. Rep., 1997.
- [4] L. S. Jennings, M. E. Fisher, K. L. Teo, and C. J. Goh, "MISER3: software for solving optimal control problems," <http://www.boeing.com/assocproducts/socs/index.html>, Tech. Rep., 1997.
- [5] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control and Dynamics*, vol. 21, p. 193, 1998.
- [6] R. E. Bellman, *Dynamic Programming*. Princeton University Press, 1957.
- [7] R. E. Bellman and S. Dreyfus, Eds., *Applied Dynamic Programming*. Prentice Hall, 1962.
- [8] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, ser. Optimization and Computation Series. Athena Scientific, 2000, vol. 1.
- [9] A. Verma, "An introduction to automatic differentiation," *Current Science*, vol. 78, no. 7, pp. 804–807, 2000.
- [10] A. Griewank, "On Automatic Differentiation," in *Mathematical Programming: Recent Developments and Applications*. Kluwer Academic Publishers, 1989, pp. 83–108.
- [11] A. Griewank, D. Juedes, and J. Utke, "Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++," *ACM Transactions on Mathematical Software*, vol. 22, no. 2, pp. 131–167, 1996.
- [12] T. F. Coleman and A. Verma, *ADMAT: an automatic differentiation toolbox for MATLAB*, Cornell University, 1998.
- [13] A. Schwartz, "Theory and implementation of numerical methods based on runge-kutta integration for solving optimal control problems," Ph.D. dissertation, U. C. Berkeley, 1996.
- [14] E. Polak, "On the use of consistent approximations in the solution of semi-infinite optimization and optimal control problems," *Math. Prog.*, vol. 62, pp. 385–415, 1993.
- [15] A. E. Bryson and Y. C. Ho, *Applied Optimal Control*. Hemisphere Publishing Corp., 1975.
- [16] K. RÖbenack, "Fast computation of derivatives in control — an approach based on automatic differentiation," in *Proceedings of the Third International DCDIS Conference*, 2003.
- [17] C. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland, "Adifor - generating derivative codes from fortran programs," *Scientific Programming*, vol. 1, no. 1, pp. 1–29, 1992.
- [18] T. F. Coleman and A. Verma, "Admit-1: Automatic differentiation and matlab interface toolbox," Department of Computer Science, Cornell University, Tech. Rep. CTC97TR271, 1997.

## APPENDIX I

### FOUR TESTING OPTIMAL CONTROL PROBLEMS

Please refer to [2].